Recognizing Partial Cubes in Quadratic Time

David Eppstein

Computer Science Department, University of California, Irvine
eppstein@uci.edu

Abstract. We show how to test whether a graph with n vertices and m edges is a partial cube, and if so how to find a distance-preserving embedding of the graph into a hypercube, in the near-optimal time bound $O(n^2)$, improving previous $O(nm)$-time solutions.

1 Introduction

A *partial cube* is an undirected and unweighted graph that admits a simple distance-labeling scheme: one can label its vertices by bitvectors in such a way that the distance between any two vertices equals the Hamming distance between the corresponding labels (Figure 1). That is, the graph can be *isometrically embedded* into a hypercube.

Graham and Pollak [10] were the first to discuss partial cubes, for an application involving communication networks. Since then, these graphs have been shown to model a large variety of mathematical systems, including the learning spaces (antimatroids) modeling allowable sequences in which a human student may learn a set of concepts [5], regions and region adjacency of hyperplane arrangements [16], and total orders, partial orders, or strict weak orders on any set of elements. Important subclasses of the partial cubes include trees [15] and median graphs [14]; Chepoi et al. [3] discuss several other classes characterized in terms of their planar embeddings, including the benzenoid graphs arising from organic chemistry [17]. Partial cubes admit more efficient algorithms than arbitrary graphs for several important problems including unweighted all-pairs shortest paths [8] and are the basis for several specialized graph drawing algorithms [6].

Since the time they were first studied, it has been of interest to recognize and label partial cubes. Djokovic [4] and Winkler [18] provided mathematical characterizations of partial cubes in terms of certain equivalence relations on the edges; their results can also be used to describe the bitvector labeling of the vertices of a partial cube, and to show that it is essentially unique when it exists. As Aurenhammer and Hagauer [1] showed, these characterizations can be translated into algorithms for recognizing these graphs in time $O(mn)$, where m and n are respectively the number of edges and vertices in the given graph.[1] Since then there has been additional work on algorithms for recognizing partial cubes [12] and special classes of partial cubes [2, 11, 13] but there has been no improvement to the $O(mn)$ time bound for this problem in general. In this paper we provide the first such improvement, showing how to solve the problem in time $O(n^2)$. There are two main ideas in our algorithm: First, we use bit parallelism to speed up the construction of a bitvector labeling of the vertices, and the corresponding partition of the edges, based on the characterizations of partial cubes by Djokovic and Winkler. And second, we adapt our previous fast all-pairs shortest paths algorithm for partial cubes so that it can handle non-partial-cube inputs and verify that the labeling we construct is valid.

Our algorithms depend on a RAM model of computation in which integers of at least $\log n$ bits may be stored in a single machine word, and in which addition, bitwise Boolean operations, comparisons, and table lookups can be performed on $\log n$-bit integers in constant time per operation. The constant-time assumption is standard in the analysis of algorithms, and any machine model that is capable of storing an address large enough to address the input to our problem has machine words with at least $\log n$ bits.

[1] Note that the time bound claimed in the title of Aurenhammer and Hagauer's paper is $O(n^2 \log n)$. However, as we discuss below, partial cubes have at most $n \log_2 n$ edges, so a time bound of this form is not an improvement on an $O(mn)$ bound.

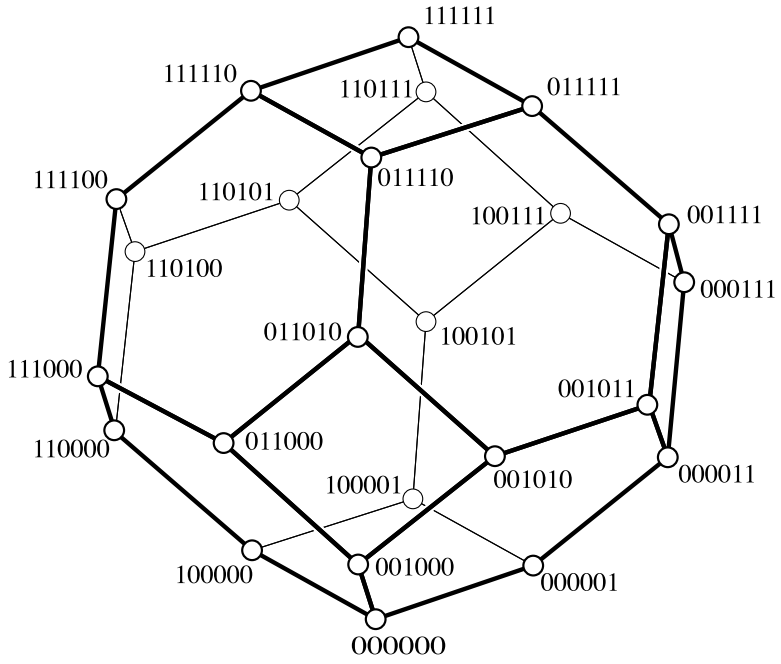

Fig. 1. A partial cube, with labeled vertices. The distance between any pair of vertices equals the Hamming distance between the corresponding labels, a defining property of partial cubes.

Our running time, $O(n^2)$, is in some sense close to optimal, as the output of the algorithm, a partial cube labeling of the input graph, may consist of $\Omega(n^2)$ bits.

2 Winkler's characterization

The characterizations of partial cubes by Djokovic [4] and Winkler [18] both depend on defining certain relations on the edges of the graph that, in the case of partial cubes, can be shown to be equivalence relations. Moreover, although Djokovic's and Winkler's relations may differ from each other on arbitrary graphs, they are identical on partial cubes. It will be more convenient for our purposes to start with the formulation of Winkler. Therefore, following Winkler, define a relation $\sim_G$ on the edges of an undirected graph G, by setting $pq \sim_G rs$ if and only if $d(p,r)+d(q,s) \neq d(p,s)+d(q,r)$, where d denotes the number of edges in the shortest path between two vertices. Note that this relation is automatically reflexive and symmetric (that is, it does not depend on the order in which the two edges are given) and it also does not depend on the ordering of the endpoints of each edges.

For example, if *pqrs* form a path, with no additional edges connecting these four vertices, then $pq \not\sim_G rs$ because $d(p,r)+d(q,s) = 2+2 = 3+1 = d(p,s)+d(q,r)$. On the other hand, if *pqrs* form a cycle, again with no additional edges, then $pq \sim_G rs$ because $d(p,r)+d(q,s) = 2+2 \neq 1+1 = d(p,s)+d(q,r)$. Figure 2 shows a more complicated example of this relation. The graph $K_{2,3}$ shown in the figure is not a partial cube, and the relation $\sim_G$ defined from it fails transitivity and therefore is not an equivalence relation.

Lemma 1 (Winkler). *Graph G is a partial cube if and only if G is bipartite and $\sim_G$ is an equivalence relation.*

We will use $[e]$ to denote the set of edges related to an edge e by $\sim_G$ (that is, in the case that G is a partial cube, the equivalence class of e). If G is a partial cube, then any subgraph $G \setminus [e]$ has two connected components. The partial cube labeling for G has a coordinate i such that the ith bit in all labels for vertices in

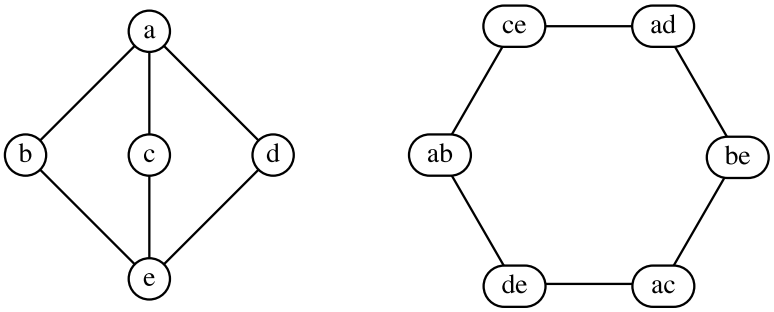

Fig. 2. An example of Winkler's relationship, for the graph $G = K_{2,3}$ (left). In this graph, each edge is related to the two other edges that it does not share an endpoint with; the right side of the figure shows pairs of edges that are related to each other. In this graph, $\sim_G$ is not an equivalence relationship; for instance, $ab \sim_G ce \sim_G ad$, but $ab \not\sim_G ad$. Thus, by Winkler's characterization, $K_{2,3}$ is not a partial cube.

one of the two components is 0, and the same bit in all labels for vertices in the other component is 1. Thus, as shown by Winkler, the dimension of the partial cube labeling equals the number of equivalence classes of $\sim_G$, and the labeling itself is essentially unique up to symmetries of the hypercube.

It will be important for our algorithms to observe that any partial cube with n vertices has at most $n \log n$ edges; see, for instance, Lemma 4 of [8].

3 Finding a single edge class

It is straightforward to construct $[pq]$ by performing two breadth first searches, one starting from p and another starting from q, using the resulting breadth first search trees to calculate all distances involving p or q, and then applying the definition of Winkler's relation $\sim_G$ to each other edge of the graph. We begin the description of our algorithm by showing how to simplify this construction: we may find $[pq]$ by an algorithm that performs only a single breadth first search. Moreover, we need not calculate any distances as part of this computation. This simplification will be an important step of our overall result, as will eventually allow us to construct multiple equivalence classes of edges simultaneously, in less time than it would take to perform each construction separately.

Our technique is based on the following observation:

Lemma 2. *Let pq be an edge in a bipartite graph G. Then $pq \sim_G rs$ if and only if exactly one of r and s has a shortest path to p that passes through q.*

Proof. If neither r nor s has such a path, then $d(q,r) = d(p,r)+1$ and $d(q,s) = d(p,s)+1$, so $d(p,r)+d(q,s) = d(p,r)+1+d(p,s) = d(q,r)+d(p,s)$ by associativity of addition, and $pq \not\sim_G rs$. Similarly, if both r and s have such paths, then $d(q,r) = d(p,r)-1$ and $d(q,s) = d(p,s)-1$, so $d(p,r)+d(q,s) = d(p,r)-1+d(p,s) = d(q,r)+d(p,s)$. Thus in neither of these cases can pq and rs be related. If, on the other hand, exactly one of r and s has such a path, we may assume (by swapping r and s if necessarily that it is r that has the path through q. Then $d(q,r) = d(p,r)-1$ while $d(q,s) = d(p,s)+1$, so $d(p,r)+d(q,s) = d(p,r)+d(p,s)+1 \neq d(p,r)-1+d(p,s) = d(q,r)+d(p,s)$, so in this case $pq \sim_G rs$. □

Thus, to find the edge class $[pq]$ in a bipartite graph G, we may perform a breadth first search rooted at p, maintaining an extra bit of information for each vertex v traversed by the search: whether v has a shortest path to p that passes through q. This bit is set to false initially for all vertices except for q, for which it is true. Then, when the breadth first search traverses an edge from a vertex v to a vertex w, such that w has not yet

been visited by the search (and is therefore farther from p than v), we set the bit for w to be the disjunction of its old value with the bit for v. Note that we perform this update for all edges of the graph, regardless of whether the edges belong to any particular breadth first search tree.

Let S_{pq} denote the set of vertices nearer to p than to q; these sets were called *semicubes* in our algorithm for lattice embeddings of partial cubes [7], where they play a key role, and they are also central to Djokovic's characterization of partial cubes. It will be important to the correctness of our algorithm to make the following additional observation.

Lemma 3. *If G is bipartite, then for any edge pq the semicubes S_{pq} and S_{qp} partition G into two subsets, and the edge class $[pq]$ forms the cut between these two semicubes.*

Proof. This follows immediately from the previous lemma, since S_{qp} consists exactly of the vertices that have a shortest path to p passing through q. □

We remark that this description of edge classes $[pq]$ in terms of semicubes is very close to Djokovic's original definition of an equivalence relation on the edges of a partial cube. Thus, for bipartite graphs, Winkler's definition (which we are following here) and Djokovic's definition can be shown to coincide.

4 Finding several edge classes

As we now show, we can apply the technique described in the previous section to find several edge classes at once. Specifically, we will find classes $[pq]$ for each neighbor q of a single vertex p, by performing a single breadth first search rooted at p.

Lemma 4. *Let pq and pr be edges in a bipartite graph G. Then $pq \not\sim_G pr$.*

Proof. By bipartiteness, $d(q,r) = 2$, so $d(p,p) + d(q,r) = 2 = 1 + 1 = d(p,r) + d(q,p)$. □

Our algorithm will need efficient data structures for storing and manipulating bit vectors, which we now describe. As described in the introduction, we assume throughout that arithmetic and bitwise Boolean operations on integers of at least $\log n$ bits, as well as array indexing operations, are possible in constant time.

Lemma 5. *Let k be a given number, and let $K = 1 + k/\log n$. Then it is possible to store bitvectors with k bits each in space $O(K)$ per bitvector, and perform disjunction operations and symmetric difference operations in time $O(K)$ per operation. In addition, in time $O(K)$ we can determine whether a bitvector contains any nonzero bits. If it does, in time $O(K)$ we can determine whether it has exactly one nonzero bit, and if so find the index of that bit, using a single precomputed external table of size n.*

Proof. We store a bitvector in $\lceil K \rceil$ words, by packing $\log n$ bits per machine word. Disjunction and symmetric difference can be performed independently on each of these words. To test whether a bitvector is nonzero, we use a comparison operation to test whether each of its words is nonzero. To test whether a bitvector has exactly one nonzero bit, and if so find out which bit it is, we again use comparisons to test whether there is exactly one word in its representation that is nonzero, and then look up that word in a table that stores either the index of the nonzero bit (if there is only one) or a flag value denoting that there is more than one nonzero bit. □

We are ready to specify the main algorithm of this section, for finding a collection of edge classes of our supposed partial cube.

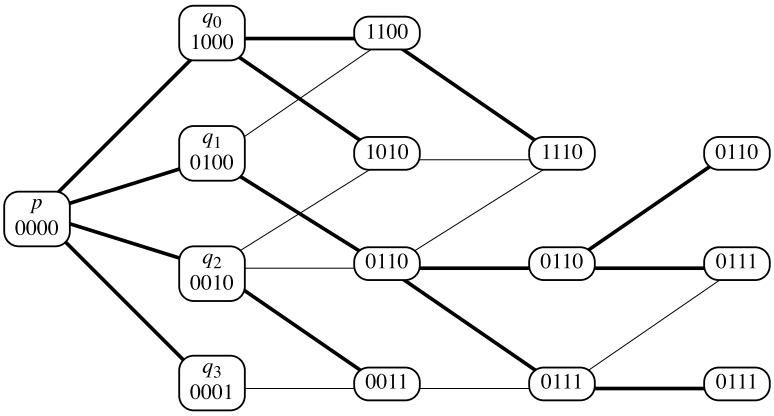

Fig. 3. The vertex-labeling stage of the algorithm of Lemma 6. The breadth first search tree edges are shown darker than the other edges; the left-to-right placement of the vertices is determined by their distance from the starting vertex p. Except for the neighbors q_i of the starting vertex, the bitvector shown for each vertex is the disjunction of the bitvectors of its neighbors to the left.

Lemma 6. *Let G be any graph with n vertices and m edges. Then there is an algorithm which either determines that G is not a partial cube (taking time at most n^2 to do so) or finds a collection $\mathcal{E}$ of disjoint sets of edges $[e_i]$, with $|E| \geq 2m/n$, taking time $O(|\mathcal{E}| \cdot n)$ to do so. In the latter case, the algorithm can also label each vertex of G by the set of semicubes it belongs to among the semicubes corresponding to the edges e_i, in the same total time.*

Proof. We first check that G is bipartite; if not, it cannot be a partial cube. We also check that its number of edges is at most $n \log_2 n$, and if not we again report that it is not a partial cube. We then let p be a vertex of maximum degree in G. We denote by d the degree of p, which must be at least $2m/n$. We denote the d neighbors of p in G by q_i, for an index i satisfying $0 \leq i < d$.

We create, for each vertex of G, a data structure D_v with d bits $D_v[i]$. Bit $D_v[i]$ will eventually be 1 if v has a shortest path to p that passes through q_i (that is, if $v \in S_{q_i p}$); initially, we set all of these bits to 0 except that we set $D_{p_i}[i] = 1$. Next, we perform a breadth first traversal of G, starting at p. When this traversal finds an edge from a vertex v to a vertex w that has not yet been traversed (so w is farther from p than v), it sets all bits $D_w[i]$ to be the disjunction of their previous values with the corresponding bits $D_v[i]$, as shown in Figure 3.

Finally, once the breadth first search is complete and all data structures D_v have reached their final values, we examine each edge vw in the graph. If $D_v = D_w$, we ignore edge vw, as it will not be part of our output collection. Otherwise, we compute a bitvector B as the symmetric difference of D_v and D_w. If B contains two or more nonzero bits $B[i]$ and $B[j]$, then vw belongs to both $[pq_i]$ and $[pq_j]$, and G cannot be a partial cube; if we ever encounter this condition we terminate the algorithm and report that the graph is not a partial cube. Otherwise, we assign vw to the class $[pq_i]$ for which $B[i]$ is nonzero. Figure 4 shows this assignment of edges to classes for the example graph shown in Figure 3.

The result of this algorithm is a collection $\mathcal{E}$ of disjoint sets of edges $[pq_i]$, as the lemma requires; the number of sets in the collection is d. All stages of the algorithm perform $O(m)$ steps, each one of which involves at most $O(1)$ of the bitvector operations described by Lemma 5, so the total time is $O(m(1 + d/\log n)) = O(d(m/d + m/\log n)) = O(dn)$, as claimed.

The semicube labeling output described by the statement of the lemma is represented by the data structures D_v computed as part of the algorithm. □

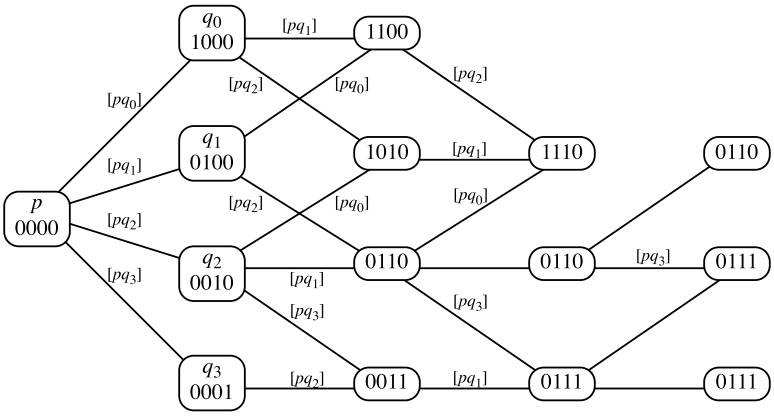

Fig. 4. The edge-labeling stage of the algorithm of Lemma 6. If the bitvectors of the endpoints of an edge differ only in their ith bits, the edge is included in class $[pq_i]$. If the bitvectors of the endpoints are the same, the edge is not included in any class. If there were an edge that had bitvectors differing in more than one bit, the graph would not be a partial cube.

5 Finding all edge classes

In order to recognize a partial cube, we need to partition its edges into equivalence classes of the relation $\sim_G$, and then verify that the resulting labeling is correct. The algorithm of the previous section allows us to find some of these equivalence classes efficiently, but as it depends for its efficiency on starting from a high degree vertex we will not necessarily be able to use it multiple times on the same graph. In order to reapply the algorithm and find all equivalence classes efficiently, as we now describe, we will need to remove from the graph the parts we have already recognized.

Lemma 7. *Let G be a partial cube, let pq be an edge in G, and let G' be the graph formed from G by contracting all edges in $[pq]$. For any edges e and f in G, neither of which belong to $[pq]$, let e' and f' denote the corresponding edges in G'. Then $e \sim_G f$ if and only if $e' \sim_{G'} f'$.*

Proof. If e and f are not in $[pq]$, by Lemma 3, either both edges connect vertices in one of the two semicubes S_{pq} and S_{qp}, or one edge is entirely in one semicube and the other edge is in the other semicube. If both are in the same semicube, then no shortest path from any vertex of e to any vertex of f can use an edge of $[pq]$ (for if it did, that crossing would increase rather than decrease the Hamming distance of the path vertex's labels), so the distances $d(x,y)$ used in the definition of $\sim_{G'}$ remain unchanged from those used to define $\sim_G$. If, on the other hand, e and f are in opposite semicubes, then by similar reasoning every shortest path from an endpoint of e to a vertex of f must use exactly one edge of $[pq]$, and each distance $d(x,y)$ used in the definition of $\sim_{G'}$ is exactly one smaller than the corresponding distance in the definition of $\sim_G$. Since we are subtracting two units of distance total from each side of the inequality by which $\sim_{G'}$ is defined, it remains unchanged from $\sim_G$. □

Lemma 8. *Let G be a partial cube, let pq be an edge in G, and let G' be the graph formed from G by contracting all edges in $[pq]$. Then G' is a partial cube, the equivalence classes of edges in G' correspond with those in G except for $[pq]$, and the vertex labeling of G' is formed by omitting the coordinate corresponding to $[pq]$ from the vertex labeling of G.*

Proof. By Lemma 7, $\sim_{G'}$ coincides with $\sim_G$ on the remaining edges; thus, it is an equivalence relation, G' is a partial cube, and its equivalence classes correspond with those of G. Since the vertex labeling is formed

from the semicubes of G', which are derived from the cuts formed by equivalence classes of edges, they also correspond in the same way. □

Lemma 9. *Any partial cube with n vertices has at most $n-1$ edge equivalence classes.*

Proof. Choose arbitrarily a vertex v. For any edge equivalence class $[pq]$, with p closer to v than q is, any shortest path from v to q must pass through an edge in $[pq]$ by Lemma 3. In particular, if T is a breadth-first spanning tree of the graph, rooted at v, T must include an edge in $[pq]$. But T has only $n-1$ edges, and each equivalence class is represented by at least one edge in T, so there can be at most $n-1$ equivalence classes. □

Our algorithm for partitioning the edges of a graph G into classes (that, if G is a partial cube, will be the equivalence classes of $\sim_G$) and simultaneously labeling the vertices of G with bitvectors (that, if G is a partial cube, will be a correct partial cube labeling for G) performs the following steps. As part of the algorithm, we set a limit L on the number of equivalence classes it can output; for our initial call to the algorithm, we set $L = n-1$, but it will be smaller in the recursive calls the algorithm makes to itself.

- If G has one vertex and no edge, we report that it is a partial cube, label its vertex with a bitvector of length zero, and return an empty set of edge equivalence classes.
- We find the maximum degree d of a vertex in G and test whether d exceeds the remaining limit on the number of allowed equivalence classes. If it does, we terminate the algorithm and report that G is not a partial cube.
- We apply the algorithm of Lemma 6 to find a set $\mathcal{E}$ of d edge classes of G. If this algorithm terminates and reports that G is not a partial cube, we do likewise.
- We contract all edges that belong to classes in $\mathcal{E}$, and remove any self-loops or multiple adjacencies in the resulting contracted graph. As we do so, we maintain a correspondence of edges in G with the edges representing them in the contracted graph G', and between vertices in G and the corresponding vertices in G'. If a set of edges in G corresponds to a multiple adjacency in G', we represent them all by the same single edge in G'. If an edge in G corresponds to a self-loop in G', and does not belong to one of the classes in $\mathcal{E}$, we terminate the algorithm and report that G is not a partial cube.
- We apply the same algorithm recursively, to partition the edges and label the vertices of G'. In this recursive call we limit the algorithm to output at most $L-d$ equivalence classes. If this algorithm terminates and reports that G' is not a partial cube, we terminate and report that G is also not a partial cube.
- We propagate the labels and partition of G' back to the vertices and edges of G, using the correspondence created when we contracted G to form G'.
- To form the list of equivalence classes of edges for G, we concatenate the list of equivalence classes for G' (with the edges replaced by the edges they correspond to in G) with the separate list of classes $\mathcal{E}$.
- To form the vertex label for each vertex v of G, we concatenate the bitvector for the vertex corresponding to v in G' with the bitvector D_v found by the algorithm of Lemma 6.

As an example, if we apply our algorithm to the graph of Figures 3 and 4 (perhaps the graph contains an additional edge, not shown, that would cause the vertex p to have maximum degree), it would construct the four edge classes and four-bit labels shown in Figure 4 in its outermost call. It would then contract the labeled edges, resulting in a much smaller graph, a path of three edges: there are four unlabeled edges in Figure 4 but two of them form a multiple adjacency when contracted. We pass this path to the second level of recursion, which will label and contract two of the edges and leave unlabeled the third since a path has no nontrivial edge relations. In the third level of recursion, the remaining edge is labeled and contracted, leaving a single vertex in the fourth level of recursion, which terminates immediately. Thus, for this graph (which is a partial cube), the algorithm eventually terminates with seven edge classes: the four shown in Figure 4, one for the two unlabeled edges that are part of a four-cycle in that figure, and one each for the two remaining edges.

Lemma 10. *The algorithm above terminates in time $O(n^2)$, and either produces a partition of the edges into classes and a bitvector labeling of the vertices or terminates with the claim that G is not a partial cube. If G is a partial cube, the algorithm produces a correct partition and a correct labeling of G. If G is not a partial cube, but the algorithm nevertheless returns a partition and a bitvector labeling, then each edge set in the partition forms a cut in the graph separating the vertices for which the bit corresponding to that edge set is 0 from the vertices for which the bit is 1.*

Proof. As is standard in graph algorithms, removing self-loops and multiple adjacencies from the contracted graph G' may be performed in time $O(m)$ by assigning index numbers to the vertices and then applying two rounds of bucket sorting to the list of edges, one for each endpoint of each edge. The other steps of the algorithm, except for applying Lemma 6 and concatenating vertex labels, take time $O(m)$. By Lemma 6, the time to find $\mathcal{E}$ is $O(dn)$, where d is the number of equivalence classes found. And, the time spent in the final step of the algorithm concatenating vertex labels is also $O(dn)$. Thus, in each recursive call of the algorithm, the time taken at that level of the recursion is $O(dn+m) = O(dn)$. Since we limit the algorithm to produce a total of at most $n-1$ classes, the total time summed over all recursive calls is at most $O(n^2)$.

If the input is a partial cube, we prove by induction on the number of recursive calls that the output is correct. As a base case, this is clearly true for the single-vertex graph. Otherwise, each call to the algorithm of Lemma 6 finds a valid set of classes $[pq]$, which by Lemma 1 are equivalence classes of $\sim_G$, and a valid vertex labeling for the semicubes derived from those classes. The induction hypothesis tells us that the algorithm finds a correct labeling and partitioning for the contracted graph G', and by Lemma 8 it is also correct when translated to the corresponding objects of G. The algorithm simply combines these two components of a correct labeling and therefore all equivalence classes it outputs are correct. By the induction hypothesis again, every edge of G' is part of one of the output equivalence classes, from which it follows that these classes when translated to G include all edges not already part of a class in $\mathcal{E}$; therefore our output list of equivalence classes is not only correct but complete, and forms a partition of the edges of G.

If the input is not a partial cube, the desired edge cut property nevertheless follows for the edge classes in $\mathcal{E}$ by Lemma 3, and can be shown to hold for all edge classes straightforwardly by induction on the number of recursive calls. □

6 All pairs shortest paths

In order to verify that the given graph is partial cube, we check that the labeling constructed by Lemma 10 is a correct partial cube labeling of the graph. To do this, we need distance information about the graph, which (if it is a correctly labeled partial cube) can be gathered by the all-pairs shortest paths algorithm for partial cubes from our previous paper [8]. However, as part of our verification algorithm, we will need to apply this algorithm to graphs that may or may not be partial cubes. So, both for the purpose of providing a self-contained explanation and in order to examine what the algorithm does when given an input that may not be a partial cube, we explain it again in some detail here.

It will be convenient to use some of the language of *media theory* [9], a framework for describing systems of states and actions on those states (called *media*) as finite state machines satisfying certain axioms. The states and adjacent pairs of states in a medium form the vertices and edges of a partial cube, and conversely any partial cube can be used to form a medium. We do not describe here the axioms of media theory, but only borrow sufficient of its terminology to make sense of the all-pairs shortest path algorithm.

Thus, we define a *token* to be an ordered pair of complementary semicubes (S_{pq}, S_{qp}). If G is a graph, with vertices labeled by bitvectors, we may specify a token as a pair (i,b) where i is the index of one of the coordinates of the bitvectors, S_{pq} is the semicube of vertices with ith coordinate equal to b, and S_{qp} is the semicube of vertices with ith coordinate unequal to b. A token *acts* on a vertex v if v belongs to S_{pq} and has a neighbor w in S_{qp}; in that case, the result of the action is w. Our all-pairs shortest path algorithm begins by

building a table indexed by (vertex,token) pairs, where each table cell lists the result of the action of a token τ on a vertex v (or v itself if τ does not act on v). Note that, if we are given any labeled graph that may or may not be a correctly labeled partial cube, we may still build such a table straightforwardly in time $O(n^2)$; if as part of this construction we find that a vertex v has two or more neighbors in S_{qp} we may immediately abort the algorithm as in this case the input cannot be a correctly labeled partial cube.

Define an *oriented tree rooted at r* to be a subgraph of the input graph G, with an orientation on each edge, such that each vertex of G except for r has a single outgoing edge vw, and such that w is formed by the action on v of a token (S_{pq}, S_{qp}) for which r is a member of S_{qp}.

Lemma 11. *Suppose we are given a graph G, a labeling of the vertices of G by bitvectors, and a partition of the edges into classes, such that each class is the set of edges spanning the cut defined by one of the coordinates of the bitvectors. Then the graph distance between any two vertices v and w in G is greater than or equal to the Hamming distance of the labels of v and w.*

Proof. For each bit in which the labels of v and w differ, the path from v to w must cross the corresponding cut in G at least once. No two cuts can share the same path edge, as the cuts partition the edges. Therefore, any path from v to w must have at least as many edges as there are bit differences. □

Lemma 12. *Suppose we are given a graph G, a labeling of the vertices of G by bitvectors, and a partition of the edges into classes, such that each class is the set of edges spanning the cut defined by one of the coordinates of the bitvectors, and suppose that T is an oriented tree rooted at r. Then T is a shortest path tree for paths to r in G, and each path from any vertex s to r in this tree has length equal to the Hamming distance between the labels of s and r.*

Proof. T has no directed cycles, for traversing a cycle would cross the same cut in G multiple times in alternating directions across the cut, while in T any directed path can only cross a cut in the direction towards r. Thus, T is a tree. The length of a path in T from s to r at most equals the Hamming distance between the labels of s and r, because by the same reasoning as above the path can only cross once the cuts separating s and r (for which the corresponding bits differ) and cannot cross any cut for which the corresponding bits of the labels of s and r agree. By Lemma 11 any path must have length at least equal to the Hamming distance, so the paths in T are shortest paths and have length equal to the Hamming distance. □

Our all-pairs shortest path algorithm traverses an Euler tour of a spanning tree of the input graph, making at most $2n-1$ steps before it visits all vertices of the graph, where each step replaces the currently visited node in the traversal by a neighboring node. As it does so, it maintains the following data structures:

- The current node visited by the traversal, r.
- A doubly-linked ordered list L of the tokens (S_{pq}, S_{qp}) for which r belongs to S_{qp}.
- A pointer p_v from each vertex $v \neq r$ to the first token in L that acts on v.
- A list A_τ for each token τ in L of the vertices pointing to i.

Lemma 13. *If the data structures described above are maintained correctly, we can construct an oriented tree rooted at r.*

Proof. We set the directed edge out of each v to be the result of the action of token p_v on v. □

To update the data structure when traversing from r to r', we perform the following steps:

- Append the token $\tau = (S_{rr'}, S_{r'r})$ to the end of L, set $p_r = \tau$, and add r to A_τ.
- Let τ' be the token $(S_{r'r}, S_{rr'})$; remove r' from $A_{\tau'}$.
- For each vertex $v \neq r$ in $A_{\tau'}$, search L sequentially forward from τ' for the next token that acts on v. Replace p_v with a pointer to that token and update the lists A_i appropriately.

– Remove $(S_{r'r}, S_{rr'})$ from L.

We modify the algorithm in one small regard to handle the possibility that the input might not be a partial cube: if the search for the replacement for p_v runs through all of list L without finding any token that acts on v, we abort the algorithm and declare that the input is not a partial cube.

Lemma 14. *If the input graph G is a correctly labeled partial cube, the algorithm described above will correctly update the data structures at each step and find a shortest path tree rooted at each node. If the input graph is not a correctly labeled partial cube, but is a bitvector-labeled graph together with a partition of the edges into classes such that each class is the set of edges spanning the cut defined by one of the coordinates of the bitvectors, then the algorithm will abort and declare that the input is not a partial cube. In either case, the total running time is at most $O(n^2)$.*

Proof. If the input is a partial cube, then, at any step of the algorithm, each vertex v has a token in L that acts on it, namely the token corresponding to the first edge in a shortest path from v to r. Thus, the sequential search for a replacement for p_v, starting from a point in L that is known to be earlier than all tokens acting on v, is guaranteed to find such a token. Thus, by Lemma 13 we have an oriented tree rooted at r for each r, and by Lemma 12 this is a shortest path tree.

Conversely, if the algorithm terminates with an oriented tree rooted at r for each r, this gives us by Lemma 12 a shortest path tree in which each path length equals the Hamming distance of labels; since all graph distances equal the corresponding Hamming distances, the input is a partial cube. Thus, if the input were not a correctly-labeled partial cube, but satisfied the other conditions allowing us to apply Lemma 12, the algorithm must at some point abort.

L starts with at most $n-1$ items on it, and has at most $2n-1$ items added to it over the course of the algorithm. Thus, for each v, over the course of the algorithm, the number of steps performed by searching for a new value for p_v is at most $3n-2$. Thus, the total number of time spent searching for updated values of p_v is $O(n(3n-2)) = O(n^2)$. The other steps of the algorithm are dominated by this time bound. □

7 Testing correctness of the labeling

We now put together the pieces of our partial cube recognition algorithm.

Lemma 15. *If we are given a graph G, a labeling of the vertices of G by bitvectors, and a partition of the edges into classes, such that each class is the set of edges spanning the cut defined by one of the coordinates of the bitvectors, then we can determine whether the given labeling is a valid partial cube labeling in time $O(n^2)$.*

Proof. We apply the algorithm of Lemma 14. By that Lemma, that algorithm either successfully finds a collection of shortest path trees in G, which can only happen when the input is a partial cube, or it aborts and declares that the input is not a partial cube. We use the presence or absence of this declaration as the basis for our determination of whether the given labeling is valid. □

Theorem 1. *Let G be an undirected graph with n vertices. Then we may check whether G is a partial cube, and if so construct a valid partial cube labeling for G, in time $O(n^2)$.*

Proof. We use Lemma 10 to construct a partial cube labeling, and Lemma 15 to test its validity. □

We observe that this is nearly optimal because (for instance, in the case of a tree) the labeling produced by the algorithm may consist of $\Omega(n^2)$ bits. However, in our computational model, such a labeling may be represented in $O(n^2/\log n)$ words of storage, so the trivial lower bound on the runtime of our checking algorithm is $\Omega(n^2/\log n)$. Additionally, in the case of partial cubes with few edge equivalence classes, or other forms of output than an explicit bitvector labeling of the vertices, even faster runtimes are not ruled out. We leave any further improvements to the running time of partial cube recognition as an open problem.

References

1. F. Aurenhammer and J. Hagauer. Recognizing binary Hamming graphs in $O(n^2 \log n)$ time. *Mathematical Systems Theory* 28:387–395, 1995.
2. B. Brešar, W. Imrich, and S. Klavžar. Fast recognition algorithms for classes of partial cubes. *Discrete Applied Mathematics* 131(1):51–61, 2003.
3. V. Chepoi, F. Dragan, and Y. Vaxès. Center and diameter problems in plane triangulations and quadrangulations. *Proc. 13th ACM-SIAM Symp. Discrete Algorithms (SODA 2002)*, pp. 346–355, January 2002.
4. D. Z. Djokovic. Distance preserving subgraphs of hypercubes. *J. Combinatorial Theory, Ser. B* 14:263–267, 1973.
5. J.-P. Doignon and J.-C. Falmagne. *Knowledge Spaces*. Springer-Verlag, 1999.
6. D. Eppstein. Algorithms for drawing media. *Proc. 12th Int. Symp. Graph Drawing (GD 2004)*, pp. 173–183. Springer-Verlag, Lecture Notes in Computer Science 3383, 2004, arXiv:cs.DS/0406020.
7. D. Eppstein. The lattice dimension of a graph. *Eur. J. Combinatorics* 26(5):585–592, July 2005, http://dx.doi.org/10.1016/j.ejc.2004.05.001, arXiv:cs.DS/0402028.
8. D. Eppstein and J.-C. Falmagne. Algorithms for media. ACM Computing Research Repository, June 2002, arXiv:cs.DS/0206033.
9. J.-C. Falmagne and S. V. Ovchinnikov. Media theory. *Discrete Applied Mathematics* 121(1–3):103–118, September 2002.
10. R. L. Graham and H. Pollak. On addressing problem for loop switching. *Bell System Technical Journal* 50:2495–2519, 1971.
11. J. Hagauer, W. Imrich, and S. Klavžar. Recognizing median graphs in subquadratic time. *Theoretical Computer Science* 215:123–136, 1999.
12. W. Imrich and S. Klavžar. A simple $O(mn)$ algorithm for recognizing Hamming graphs. *Bull. Inst. Combin. Appl.* 9:45–56, 1993.
13. W. Imrich, S. Klavžar, and H. M. Mulder. Median graphs and triangle-free graphs. *SIAM J. Discrete Math.* 12:111–118, 1999.
14. H. M. Mulder. The structure of median graphs. *Discrete Mathematics* 24:197–204, 1978.
15. S. V. Ovchinnikov. The lattice dimension of a tree. arXiv.org, February 2004, arXiv:math.CO/0402246.
16. S. V. Ovchinnikov. Hyperplane arrangements in preference modeling. *J. Mathematical Psychology*, 2005. In press.
17. O. E. Polansky and D. H. Rouvray. Graph-theoretical treatment of aromatic hydrocarbons. II. The analysis of all-benzenoid systems. *Informal Communications in Mathematical Chemistry, Match No. 2*, pp. 111–115. Inst. für Strahlenchemie, Max-Planck-Institut für Kohlenforschung, Mülhein a.d. Ruhr, 1976.
18. P. Winkler. Isometric embeddings in products of complete graphs. *Discrete Applied Mathematics* 7:221–225, 1984.